Lezione 6: macchine, linguaggi, funzioni

Lezione del 23/03/2023

Facciamo il punto

- Siamo partiti cercando di capire come risolvere automaticamente i problemi
- E abbiamo studiato la soluzione proposta da Alan Turing che, partendo dalla sua analisi del processo di soluzione, è arrivato a definire il passo elementare di calcolo: una operazione
  - scelta in un insieme di operazioni di cardinalità costante
  - che richiede di ricordare una quantità costante di dati
- Da questa idea di operazione elementare, Turing ha introdotto un modello di calcolo: la Macchina di Turing
  - che è un linguaggio per descrivere algoritmi
  - e ogni macchina di Turing è un algoritmo
- Poi, Turing ha anche progettato la sua macchina Universale – ma questa è un'altra storia (che già conosciamo)

A questo punto

- Beh, a questo punto è ragionevole porsi un po' di domande:
  - utilizzando la Macchina di Turing, possiamo risolvere ***tutti*** i problemi? Oppure esiste qualche problema che non è risolubile con la Macchina di Turing?
  - E, se esiste qualche qualche problema che non è risolubile con la Macchina di Turing, non sarà forse possibile risolvere quel problema con un altro modello di calcolo?
- La prima domanda cui risponderemo è la seconda
- Prima di farlo, dobbiamo, però, essere un po' più precisi
- Meglio: dobbiamo essere più formali
- Siamo alla dispensa 3, paragrafo 3.1

Più in dettaglio

- Una macchina di Turing (di tipo riconoscitore) è un oggetto che, se gli diamo un certo input, quella ci risponde se quell'input soddisfa una certa proprietà
- e l'input di una macchina di Turing è una parola (scritta con i caratteri di un certo alfabeto).
- Quindi: una macchina di Turing (di tipo riconoscitore) è un oggetto che, se gli scriviamo una certa parola sul nastro, quella ci risponde se quella parola soddisfa una certa proprietà
- Allora, possiamo considerare *l'insieme di tutte le parole che soddisfano quella certa proprietà* e dire: "la nostra macchina di Turing sa riconoscere le parole che appartengono a tale insieme!"
- Ma non è abbastanza formale: che vuol dire **esattamente** *riconoscere*?

Decidere un linguaggio

- Dato un alfabeto Σ, un **linguaggio** L è un insieme di parole costituite di caratteri di Σ: ossia, $L \subseteq \Sigma^*$
- Un linguaggio L è ***deciso*** da una macchina di Turing T se
  - per ogni $x \in L$, la computazione T(x) termina in q_A
  - per ogni $x \notin L$, la computazione T(x) termina in q_R
- Quindi, **le computazioni della macchina T che decide L terminano sempre**: sia che sul nastro di T venga scritto un input appartenente ad L, sia che sul nastro di T venga scritto un input non appartenente ad L, T giunge ad una conclusione
- Ossia, **T è sempre in grado di distinguere fra le parole di L e le parole che non sono in L.**
- **Qualunque sia x in Σ*, T ci dice se x è in L oppure no**

Decidere un linguaggio - esempio

- Prendiamo la macchina T_{PAL} che abbiamo visto la scorsa lezione (con le due quintuple che rigettano se la parola in input ha lunghezza dispari):
  - $\langle q_0, a, \square, q_a, D\rangle$, $\langle q_0, b, \square, q_b, D\rangle$,
  - $\langle q_a, a, a, q_a, D\rangle$, $\langle q_a, b, b, q_a, D\rangle$, $\langle q_b, a, a, q_b, D\rangle$, $\langle q_b, b, b, q_b, D\rangle$,
  - $\langle q_a, \square, \square, q_{a1}, S\rangle$, $\langle q_b, \square, \square, q_{b1}, S\rangle$,
  - $\langle q_{a1}, a, \square, q_2, S\rangle$, $\langle q_{a1}, b, b, q_R, F\rangle$, $\langle q_{b1}, a, a, q_R, F\rangle$, $\langle q_{b1}, b, \square, q_2, S\rangle$,
  - $\langle q_2, a, a, q_2, S\rangle$, $\langle q_2, b, b, q_2, S\rangle$, $\langle q_2, \square, \square, q_0, D\rangle$,
  - $\langle q_0, \square, \square, q_A, F\rangle$,
  - **$\langle q_{a1}, \square, \square, q_R, F\rangle$, $\langle q_{b1}, \square, \square, q_R, F\rangle$** .
- Ebbene, **T_{PAL} decide il linguaggio L_{PPAL}** (Pari e PALindrome) seguente:

$$L_{PPAL} = \{ x_1x_2\ldots x_{2n} \in\{a,b\}^*: n \in \mathbb{N} \ \wedge \forall i \in \{1, 2, \ldots , n\} \ [x_i = x_{2n-i+1}] \}$$

Accettare un linguaggio

- Dato un alfabeto Σ, un linguaggio L è un insieme di parole costituite di caratteri di Σ: ossia, $L \subseteq \Sigma^*$
- Un linguaggio L è ***accettato*** da una macchina di Turing T se
  - per ogni $x \in L$, la computazione T(x) termina in q_A
  - per ogni $x \notin L$, la computazione T(x) ***non*** termina in q_A
- Quindi, se sul nastro di T viene scritto un input x appartenente ad L, siamo certi (a) che T(x) termina e (b) che T(x) termina in q_A
- Se, invece, sul nastro di T viene scritto un input x non appartenente ad L, possiamo solo essere certi che T(X) non termina in q_A
- Ma, se x non appartiene ad L,
  - non è detto che T(x) termini in q_R
  - potrebbe anche non terminare
- Ossia, T è solo in grado di di dirci che una parola appartiene a L – quando questo accade!

Accettare un linguaggio - esempio

- Modifichiamo le ultime due quintuple della macchina T_{PAL} per ottenere la macchina T_{PAL1} seguente
  - $\langle q_0, a, \square, q_a, D\rangle$, $\langle q_0, b, \square, q_b, D\rangle$,
  - $\langle q_a, a, a, q_a, D\rangle$, $\langle q_a, b, b, q_a, D\rangle$, $\langle q_b, a, a, q_b, D\rangle$, $\langle q_b, b, b, q_b, D\rangle$,
  - $\langle q_a, \square, \square, q_{a1}, S\rangle$, $\langle q_b, \square, \square, q_{b1}, S\rangle$,
  - $\langle q_{a1}, a, \square, q_2, S\rangle$, $\langle q_{a1}, b, b, q_R, F\rangle$, $\langle q_{b1}, a, a, q_R, F\rangle$, $\langle q_{b1}, b, \square, q_2, S\rangle$,
  - $\langle q_2, a, a, q_2, S\rangle$, $\langle q_2, b, b, q_2, S\rangle$, $\langle q_2, \square, \square, q_0, D\rangle$,
  - $\langle q_0, \square, \square, q_A, F\rangle$,
  - $\boldsymbol{\langle q_{a1}, \square, \square, q_{a1}, F\rangle}$, $\boldsymbol{\langle q_{b1}, \square, \square, q_R, F\rangle}$.
- Ebbene, T_{PAL1} accetta il linguaggio L_{PPAL} ma non lo decide; in particolare:
  - accetta le parole palindrome di lunghezza pari
  - rigetta le parole non palindrome
  - rigetta le parole palindrome di lunghezza dispari che hanno ‘b’ come carattere centrale
  - non termina sulle parole palindrome di lunghezza dispari che hanno ‘a’ come carattere centrale

Linguaggi decidibili / accettabili

- Un linguaggio $L \subseteq \Sigma^*$ è **decidibile** se esiste una macchina di Turing T che lo decide
  - ossia, che, **per ogni** $x \in \Sigma^*$, T(x) termina:

    se $x \in L$ allora T(x) termina in q_A, se $x \notin L$ allora T(x) termina in q_R ,
- Quando un linguaggio L è deciso da una macchina T scriviamo: **L= L(T)**
- Un linguaggio $L \subseteq \Sigma^*$ è **accettabile** se esiste una macchina di Turing T che lo accetta
  - ossia, che, **per ogni** $x \in L$, T(x) termina in q_A,
  - se $x \notin L$ allora sappiamo solo che **T(x) non termina in q_A** : potrebbe terminare in q_R oppure non terminare
  - ricordate la storia delle istanze negative?
- Naturalmente, ogni linguaggio decidibile è anche accettabile – ma non viceversa!
  - non devo spiegarvi perché, spero…

Chiariamoci un po' le idee…

- Consideriamo il il linguaggio L_{PPAL} (Pari e PALindrome) visto poc'anzi
  - abbiamo visto la macchina T_{PAL} che lo decide
  - ma abbiamo visto anche la macchina T_{PAL1} che lo accetta senza deciderlo
- Insomma, L_{PPAL} è un linguaggio decidibile oppure no????
- Certo che è decidibile! Infatti, **esiste** una macchina che lo decide: la macchina T_{PAL} !!!!
  - **esiste**: vuol dire che basta che ce ne sia una!

Linguaggi complemento

- Dunque, mentre una macchina che decide un linguaggio su un alfabeto Σ sa ben comportarsi con tutte le parole in Σ^*
  - per ogni parola in Σ^* sa se accettare o rigettare
- una macchina che accetta un linguaggio su un alfabeto Σ, invece, non sa sempre come comportarsi sulle parole in Σ^* che non sono in L
  - potrebbe esistere una parola in $\Sigma^*- L$ sulla quale la macchina non riesce a capire che decisione prendere – e quindi non termina
- Sia $L \subseteq \Sigma^*$; chiamiamo ***linguaggio complemento*** di L il linguaggio $\mathbf{L^C= \Sigma^* - L}$
- Allora, possiamo dire che *la differenza fra decisione e accettazione di un linguaggio è il comportamento della macchina sul linguaggio complemento*
  - eccole ancora qui, le istanze negative...

Teorema 3.1

- **$L \subseteq \Sigma^*$ è decidibile se e soltanto se L è accettabile e L^C è accettabile**

- Se L è decidibile, allora:
  - chiamiamo T la macchina che decide L
  - dobbiamo costruire una macchina **T_1** che accetta L e una macchina **T_2** che accetta L^C
  - Ebbene: la macchina **T_1** è la stessa macchina T
    - infatti, **per ogni $x \in L$**, T(x) termina in q_A,
  - E la macchina **T_2** ?
  - Facile: prendiamo T, invertiamo i suoi stati di accettazione e di rigetto e otteniamo **T_2**
    - infatti, poiché T decide L
    - allora **per ogni $x \notin L$**, T(x) termina in q_R,
    - ossia, **per ogni $x \in L^C$**, T(x) termina in q_R,
    - e, dunque, **per ogni $x \in L^C$**, **T_2**(x) termina in q_A!

Teorema 3.1

- **$L \subseteq \Sigma^*$ è decidibile se e soltanto se L è accettabile e L^C è accettabile**

- Se L è accettabile e L^C è accettabile allora (fate sempre riferimento alla dispensa 3, pag. 3):
  - chiamiamo T_1 la macchina che accetta L e T_2 la macchina che accetta L^C
  - dobbiamo costruire una macchina T che decide L
  - dotiamo T di due nastri: T usa il nastro 1 per *simulare* $T_1(x)$ e il nastro 2 per simulare $T_2(x)$.
  - **Input di T**: una parola x scritta sul nastro 1
  - **Inizializzazione**: T copia l'input x sul nastro 2, e poi inizia la computazione vera e propria:
    - 1) T simula **un passo** di $T_1(x)$: se quel passo fa accettare T_1 allora accetta, altrimenti va a 2)
    - 2) T simula **un passo** di $T_2(x)$: se quel passo fa accettare T_2 allora rigetta, altrimenti va a 1)
- poiché $x \in L$ oppure $x \notin L$, allora, prima o poi T_1 accetta o T_2 accetta: allora, T decide L.
- Ma perché simuliamo un passo alla volta di ciascuna macchina?! Perché non simuliamo prima l'intera $T_1(x)$ e poi l'intera $T_2(x)$?

Perché un passo alla volta?

- La macchina T_{PAL1} che abbiamo visto poc'anzi, accetta L_{PPAL}
- la seguente macchina, che chiamiamo T_{PAL2}, accetta : $L_{PPAL}{}^{C}$
  - $\langle q_0, a, \square, q_a, D \rangle$, $\langle q_0, b, \square, q_b, D \rangle$,
  - $\langle q_a, a, a, q_a, D \rangle$, $\langle q_a, b, b, q_a, D \rangle$, $\langle q_b, a, a, q_b, D \rangle$, $\langle q_b, b, b, q_b, D \rangle$,
  - $\langle q_a, \square, \square, q_{a1}, S \rangle$, $\langle q_b, \square, \square, q_{b1}, S \rangle$,
  - $\langle q_{a1}, a, \square, q_2, S \rangle$, $\langle q_{a1}, b, b, q_A, F \rangle$, $\langle q_{b1}, a, a, q_A, F \rangle$, $\langle q_{b1}, b, \square, q_2, S \rangle$,
  - $\langle q_2, a, a, q_2, S \rangle$, $\langle q_2, b, b, q_2, S \rangle$, $\langle q_2, \square, \square, q_0, D \rangle$,
  - $\langle q_0, \square, \square, q_R, F \rangle$,
  - $\langle q_{a1}, \square, \square, q_A, F \rangle$, $\langle q_{b1}, \square, \square, q_A, F \rangle$.
- Ora, costruiamo la macchina T'_{PAL} che ha due nastri: dopo aver copiato l'input x (che inizialmente è scritto sul nastro 1) sul nastro 2, T usa il nastro 1 per simulare $T_{PAL1}(x)$ e il nastro 2 per simulare $T_{PAL2}(x)$

Perché un passo alla volta?

- Costruiamo la macchina T'_{PAL} che opera in due fasi:
  - durante la prima fase simula *l'intera computazione* T_{PAL1}(aba)
  - durante la seconda fase simula *l'intera computazione* T_{PAL2}(aba)
- Bene. Ora eseguiamo la computazione T'_{PAL}(bab)
  - che, ad un certo punto, dovrà eseguire la quintupla **$\langle q_{a1}, \square, \square, q_{a1}, F \rangle$** - e, quindi, andrà in loop!
- Osservate che aba ∈ L_{PPAL}^{C}: quindi, T'_{PAL}(aba) dovrebbe rigettare
- Ma aba è una parola palindroma di lunghezza dispari con 'a' al centro
- e, quindi, poiché T'_{PAL} simula <u>prima</u> *l'intera computazione* T_{PAL1}(bab), T'_{PAL} non termina!
- Ecco perché "un passo alla volta"!
- Guardatelo bene, questo esempio

Esercizi

- E ora un paio di esercizi (<u>che vi chiedo all'esame</u>):
  - dimostrare che, se L_1 e L_2 sono due linguaggi accettabili, allora $L_1 \cup L_2$ è accettabile
  - dimostrare che, se L_1 e L_2 sono due linguaggi accettabili, allora $L_1 \cap L_2$ è accettabile
- In una delle due dimostrazioni è possibile prima simulare l'intera computazione di una macchina e poi l'intera computazione della seconda macchina: in quale dimostrazione?
  - la soluzione la trovate sulle dispense
  - e io sono sempre disponibile a correggere le soluzioni che vorrete inviarmi

Funzioni calcolabili

- Torniamo, per un momento, ai cari vecchi trasduttori: macchine di Turing dotate di nastro di output, che sanno calcolare il valore di una funzione generica – scrivendo tale valore sul nastro di output e terminando la computazione nello stato q_F
- Bella cosa, questa, di una macchina che sa *calcolare* il valore di una funzione – bella, sì, ma che vuol dire?
- Facile! ("La so, la so", starete gridando tutti, agitandovi sulle vostre sedie). Cioè, ad esempio:
  - $f(n) = n^2$ nel punto $n = 5$, vale 25 – ossia, $f(5) = 25$
  - $f(n) = 2^n$ nel punto $n = 9$ vale 512 - ossia, $f(9)=512$
  - facile!
  - E $f(n) = \frac{1}{n-4}$ nel punto $n = 4$ vale … Ops!

Funzioni calcolabili

- Già, non è proprio così banale come sembrava…
- Allora, intanto ci limitiamo a considerare funzioni "discrete" – ossia, dati due alfabeti (finiti, neanche a dirlo) Σ_1 e Σ_2, noi consideriamo funzioni f : $\Sigma_1^* \rightarrow \Sigma_2^*$
  - ossia, funzioni che trasformano parole in altre parole
- Poi, noi vogliamo calcolarle solo dove sono definite
  - E, poiché f(n) = $\frac{1}{n-4}$ non è definita nel punto n = 4, non possiamo (né vogliamo!) calcolare f(4)
- E, infatti, parliamo di funzioni in generale (che possono non essere definite in alcuni punti) e di funzioni totali (che sono definite per ogni x ∈ Σ_1^*)
- Una funzione f : $\Sigma_1^* \rightarrow \Sigma_2^*$ è **calcolabile** se esiste una macchina di Turing di tipo trasduttore T tale che, per ogni x ∈ Σ_1^* tale che f(x) è definita, T(x)=f(x)
  - ossia, quando f(x) è definita, la computazione T(x) termina con la parola f(x) scritta sul nastro di output

Funzioni calcolabili

- Una funzione f : $\Sigma_1^* \rightarrow \Sigma_2^*$ è calcolabile se esiste una macchina di Turing di tipo trasduttore T tale che, per ogni x ∈ Σ_1^* tale che f(x) è definita, T(x)=f(x)
  - ossia, la computazione T(x) termina con la parola f(x) scritta sul nastro di output
- Osservate che questa definizione nulla ci dice circa le computazioni T(x) tali che f(x) non è definita
  - in questo caso, T(x) potrebbe non terminare
  - oppure terminare con un valore scritto sul nastro di output che non corrisponde al valore f(x): infatti, f(x) non esiste!
- Perciò, a pensarci bene, il concetto di calcolabilità di una funzione è molto simile al concetto di accettabilità di un linguaggio
- Le cose vanno certamente bene quando scegliamo un x tale che f(x) è definita / x appartiene al linguaggio
- Può succedere di tutto quando quando scegliamo un x tale che f(x) non è definita / x non appartiene al linguaggio

Funzioni e linguaggi

- Pensandoci bene, ad ogni linguaggio $L \subseteq \Sigma^*$ possiamo associare una funzione - quella che si chiama *funzione caratteristica* di un insieme: una funzione $\chi_L : \Sigma^* \rightarrow \{0,1\}$ tale che, per ogni $x \in \Sigma^*$,
  - $\chi_L(x)=1$ se $x \in L$,
  - $\chi_L(x)=0$ se $x \notin L$
- Osservate che, qualunque sia L, χ_L è una funzione <u>totale</u>

- **TEOREMA 3.2:** χ_L è calcolabile se e solo se L è decidibile
- è questo il Teorema 3.2 e la dimostrazione (facile facile facile) ve la <u>studiate sulle dispense</u>
- è argomento di esame
  - perciò, studiatelo e se avete dubbi fatemi delle domande!

Funzioni e linguaggi

- Ri-pensandoci bene, anche ad ogni funzione f : $\Sigma_1^* \rightarrow \Sigma_2^*$ possiamo associare un linguaggio $L_f \subseteq \Sigma_1^* \times \Sigma_2^*$: $L_f = \{ (x, y) \in \Sigma_1^* \times \Sigma_2^*$ **tali che y = f(x) }**
- Osservate bene: il linguaggio è costituito da <u>coppie di parole</u>
  - a ben guardare, L_f è, in qualche modo, il grafico della funzione f
- **TEOREMA 3.3:** Se f è calcolabile <u>e totale</u> allora L_f è decidibile
  - Idea della dimostrazione: sia T_f è il trasduttore che calcola f
  - Costruiamo il riconoscitore T per decidere L_f : T ha tre nastri – sul primo nastro è scritto l'input x, sul secondo nastro è scritto l'input y, il terzo nastro è un nastro di lavoro
  - T opera in due fasi:
    - FASE 1: T simula T_f(x) scrivendo il risultato f(x) = z sul terzo nastro
    - FASE 2: T confronta z con y, accettando se sono uguali rigettando se sono diverse
  - La dimostrazione che T effettivamente decide L_f è sulla dispensa: studiatela!
- Domandina: possiamo dire qualcosa su L_f se f è calcolabile ma non totale? Provate a giocarci un po'…

Funzioni e linguaggi

- **TEOREMA 3.4:** Se L_f è decidibile allora f è calcolabile
- e qui qualche commento è d'uopo
- Sappiamo che L_f è decidibile (la nostra ipotesi); allora esiste un riconoscitore T_L che, se gli scrivo sul nastro le **due parole x e y** quello, dopo un po', mi risponde "(x,y) è in L_f" (q_A) oppure "(x,y) non è in L_f"(q_R)
- Dobbiamo sfruttare T_L per costruire un trasduttore T_f che calcoli f
  - ossia, ogni volta che scrivo x (**x soltanto, x nudo e crudo**) sul nastro di T_f quello, dopo un po' termina con la parola f(x) scritta sul nastro di output
- Problema: se a T_f posso comunicare soltanto x, come faccio ad utilizzare T_L che ha bisogno di due input, x e y, per lavorare? **Chi me lo dà y???**
- Risposta: nessuno, me lo dà. Me lo devo costruire da me…
  - o meglio, devo ***enumerare*** tutti gli y possibili e provarli uno per uno!
- E allora...

Funzioni e linguaggi

- Costruiamo una macchina T_f , con 4 nastri ed un nastro di output, che opera come segue
  - inizialmente, l'input x è scritto sul primo nastro, e T_f scrive 0 sul secondo nastro
  - T_f scrive sul terzo nastro tutte le parole di lunghezza 0: ossia, la parola vuota -
  - T_f simula la computazione T_L(x,): se T_L accetta, allora T_f scrive sul nastro di output, altrimenti (e, in questo caso T_L rigetta) passa al successivo passo 1)
  - **PASSO 1)** T_f incrementa di 1 il valore scritto sul secondo nastro
  - **PASSO 2)** T_f scrive sul terzo nastro tutte le parole in Σ_2^* la cui lunghezza è il valore scritto sul secondo nastro: ad esempio, se sul secondo nastro è scritto 2 e Σ_2 = {a,b}, allora T_f scrive sul terzo nastro le parole aa, ab, ba, bb
  - **PASSO 3)** per ogni parola y scritta sul terzo nastro, T_f simula la computazione T_L (x, y): se T_L accetta, allora T_f scrive sul y nastro di output e termina, altrimenti (e, in questo caso T_L rigetta)
    - se non ha ancora esaminato tutte le parole scritte sul terzo nastro, passa alla parola successiva
    - altrimenti, se ha esaminato tutte le parole scritte sul secondo nastro e nessuna ha indotto T_L ad accettare, torna al PASSO 1)

Funzioni e linguaggi

- Osserviamo che i passi 1), 2) e 3) terminano sempre.
- Perciò, se f è definita in x_0, allora,
  - detto n_0 il numero di caratteri di $f(x_0)$,
  - quando sul secondo nastro verrà scritto n_0, sul terzo nastro verranno scritte tutte le parole di n_0 caratteri e fra esse anche la parola $f(x_0)$ (chiamiamola y_0)
  - allora, poiché tutte le computazioni $T_L(x_0, y)$ terminano, prima o poi verrà anche eseguita la computazione $T_L(x_0, y_0)$ che terminerà in q_A: così, y_0 verrà scritto sul nastro di output di T_f e la computazione $T_f(x_0)$ terminerà
- Questo dimostra che "se f è definita in x_0, allora $T_f(x_0)$ calcola $f(x_0)$"
- Quindi, f è calcolabile.
- Ma, se f non è definita in x_0, allora non verrà mai trovata una parola y_0 tale che $T_L(x_0, y_0)$ accetta – perché T_L decide $\mathbf{L_f} = \{ (x, y) \in \boldsymbol{\Sigma_1}^* \times \boldsymbol{\Sigma_2}^*$ **tali che y = f(x)** $\}$
- e, quindi, anche se L_f è decidibile, non è detto che f sia totale.

NOTA BENE

I teoremi 3.2 – 3.3 - 3.4 sono stati enunciati (e il 3.4 discusso) molto informalmente: per non appesantire la chiacchierata, non ho mai specificato dominio e codominio delle funzioni, e alfabeto dei linguaggi.

Naturalmente, voi dovrete essere più formali: esattamente come viene fatto sulle dispense.

Perché
dovete
studiare
sulle
dispense.

Senza se e senza ma. Punto.